MRR vs MAP vs NDCG: Rank-Aware Evaluation Metrics And When To Use Them

Moussa Taifi, Ph.D.
Nov 25, 2019 · 12 min read ★

Robert Delaunay, 1913, “Premier Disque”.

The ML Metrics Trap

Reporting small improvements on inadequate metrics is a well known Machine Learning trap. Understanding the pros and cons of machine learning (ML) metrics helps build personal credibility for ML practitioners. This is done to avoid the trap of prematurely proclaiming victory. Understanding metrics used for machine learning (ML) systems is important. ML practitioners invest signification budgets to move prototypes from research to production. The central goal is to extract value from

prediction systems. Offline metrics are crucial indicators for promoting a new model to production.

In this post, we look at three ranking metrics. Ranking is a fundamental task. It appears in machine learning, recommendation systems, and information retrieval systems. I recently had the pleasure to finish an excellent recommender systems specialization: The University of Minnesota Recommendation System Specialization. This specialization is a 5 courses recsys quest that I recommend. I wanted to share how I learned to think about evaluating recommender systems. Especially when the task at hand is a ranking task.

Without too much loss of generality, most recommenders do two things. They either attempt to predict a rating of an item by a user, or generate a ranked list of recommended items per user.

> It can be hard to imagine how to evaluate a recommender system. Comparing lists of recommended items to lists of relevant items is not intuitive. Traditional tasks predict who died on the Titanic, or what breed of dog is in an ImageNet dataset. These do not emphasis rank-aware ML metrics that are central to recommender systems.

If this interests you, keep on reading as we explore the 3 most popular rank-aware metrics available to evaluate recommendation systems:

- **MRR**: Mean Reciprocal Rank
- **MAP**: Mean Average Precision
- **NDCG**: Normalized Discounted Cumulative Gain

Flat and “Rank-less” Evaluation Metrics

Accuracy metrics

When dealing with ranking tasks, prediction accuracy and decision support metrics fall short. The **prediction accuracy metrics** include the mean absolute error (**MAE**), root mean square error (**RMSE**). These focus on comparing the actual vs predicted ratings. They operate at the individual rating prediction level. If a user rated an item with 4.5 these metrics tell us how far-off are our predictions if we predicted a rating of 1.2 or 4.3.

Decision support metrics

Next in line, the **decision support** metrics include **Precision, Recall** and **F1 score.** These focus on measuring how well a recommender helps users make good decisions. They help the user to select "good" items, and to avoid "bad" items. These types of metrics start to emphasize what is important for recommendation systems. If we recommend 100 items to a user, what matters most are the items in the first 5, 10 or 20 positions. Precision is the percentage of selected elements that are relevant to the user. Its focus is recommending mostly useful stuff. Recall is the percentage of relevant i that the system selected. Its focus is not missing useful stuff. The F1 score is the combination of the two. The F1 harmonic mean is a way to balance precision and recall to get a single metric.

For our ranking task, the metrics have one major drawback. These **decision support metrics cover the entire data set**. They are not targeted to the "Top-N" recommendations. Both precision and recall are about the entire result set. To expand these metrics, precision and recall are usually outfitted with a top-n bound. This comes in the form of **Precision@N** and **Recall@N.** Interestingly, I could no find a good source that describes the **F1@N score** which would represent the harmonic mean of the P@N and R@N. Let's carry on anyway.

The modified Precision@N metric is the percentage of the "top-n" items that are good. This incorporates some level of top-n evaluation. This means that it focuses on the top recommended items. However, they are still similar to the original Precision, Recall and F1 measures. They are all primarily concerned with being good at **finding** things**.** We need metrics that emphasis being good at **finding and ranking** things.

Time to level up. Let's see how rank-aware evaluation metrics can help.

Rank-Aware Evaluation Metrics

Recommender systems have a very particular and primary concern. They need to be able to put **relevant items very high up the list of recommendations**. Most probably,

the users will not scroll through 200 items to find their favorite brand of earl grey tea. We need rank-aware metrics to select recommenders that aim at these two primary goals:

1) ***Where*** does the recommender place the items it suggests?

2) How good is the recommender at modeling ***relative preference***?

This is where the following metrics can help:

> *MRR: Mean Reciprocal Rank*
>
> *MAP: Mean Average Precision*
>
> *NDCG: Normalized Discounted Cumulative Gain*

The 3 metrics above come from two families of metrics. The first family comprises **binary relevance based metrics**. These metrics care to know if an item is good or not in the binary sense. The second family comprises **utility based metrics.** These expand the sense of good/bad with a measurement of absolute or relative goodness. Let us describe the characteristics of each metric in the following section.

MRR: Mean Reciprocal Rank

This is the simplest metric of the three. It tries to measure "Where is the first relevant item?". It is closely linked to the binary relevance family of metrics. The algorithm goes as follows:

For each user u:

- Generate list of recommendations
- Find rank k_u of its first relevant recommendation (the first rec has rank 1)
- Compute reciprocal rank $\frac{1}{k_u}$

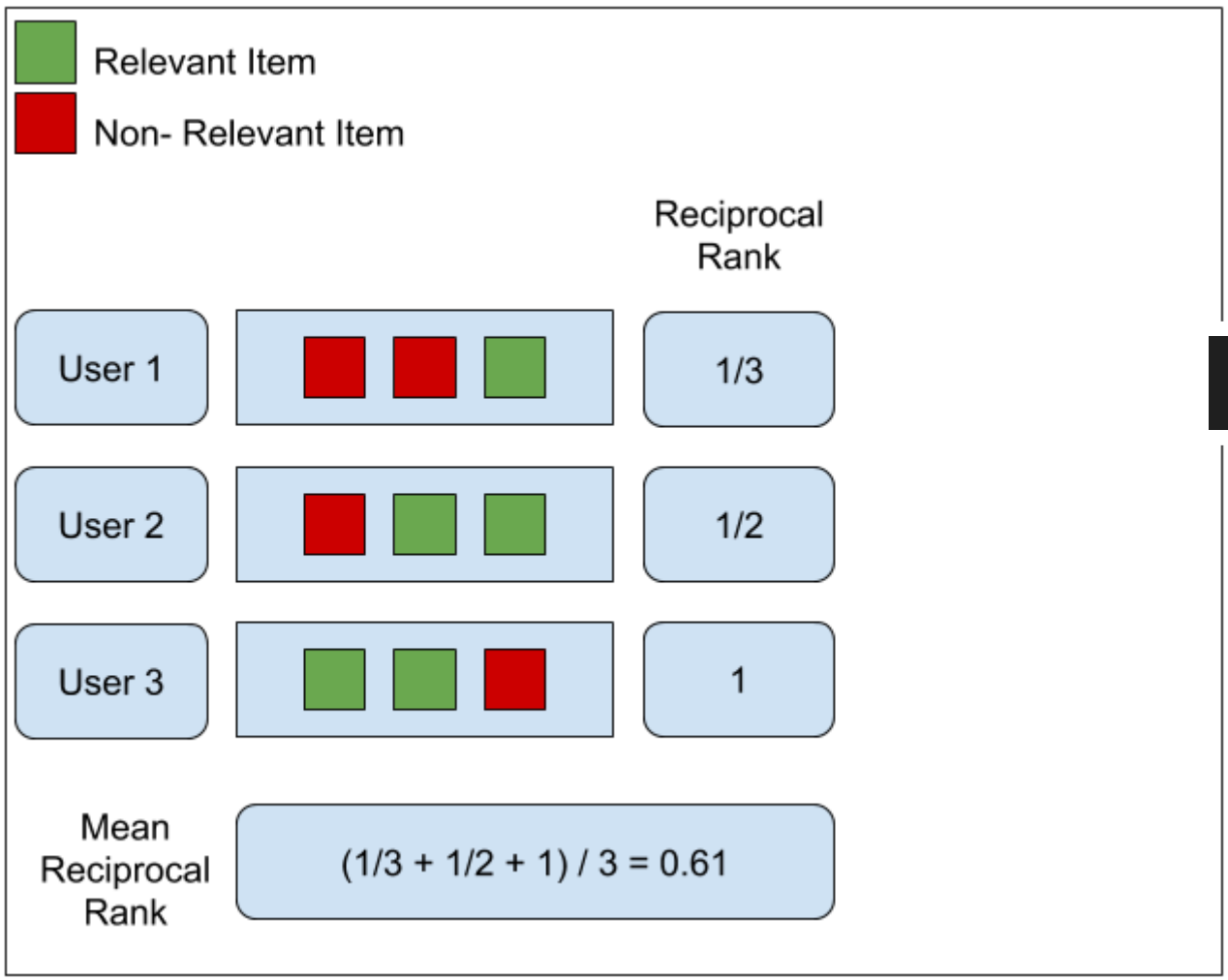
Overall algorithm performance is mean recip. rank:

$$\mathrm{MRR}(O, U) = \frac{1}{|U|} \sum \frac{1}{k_u}$$

$u \in U$

MRR metric calculation

Suppose we have the following three recommendation lists for three users. We can compute the reciprocal rank of each user by finding the rank of the first relevant item, per list. Then we do a simple averaging over all users.



Example MRR calculation

MRR Pros

- This method is simple to compute and is easy to interpret.
- This method puts a high focus on the first relevant element of the list. It is best suited for targeted searches such as users asking for the “best item for me”.
- Good for known-item search such as navigational queries or looking for a fact.

MRR Cons

- The MRR metric does not evaluate the rest of the list of recommended items. It focuses on a single item from the list.

- It gives a list with a single relevant item just a much weight as a list with many relevant items. It is fine if that is the target of the evaluation.
- This might not be a good evaluation metric for users that want a list of related items to browse. The goal of the users might be to compare multiple related items.

MAP: Average Precision and Mean Average Precision

Next is the MAP metric. Let's say we have a binary relevance data set. We want to evaluate the whole list of recommended items up to a specific cut-off N. This cut-off was previously incorporated using the Precision@N metric. The P@N decision support metric calculates the fraction of n recommendations that are good. The drawback of this metric is that it does not consider the recommended list as an **ordered list**. P@N considers the whole list as a set of items, and treats all the errors in the recommended list equally.

The goal is to cut the error in the first few elements rather than much later in the li
For this, we need a metric that weights the errors accordingly. The goal is to weight heavily the errors at the top of the list. Then gradually decrease the significance of the errors as we go down the lower items in a list.

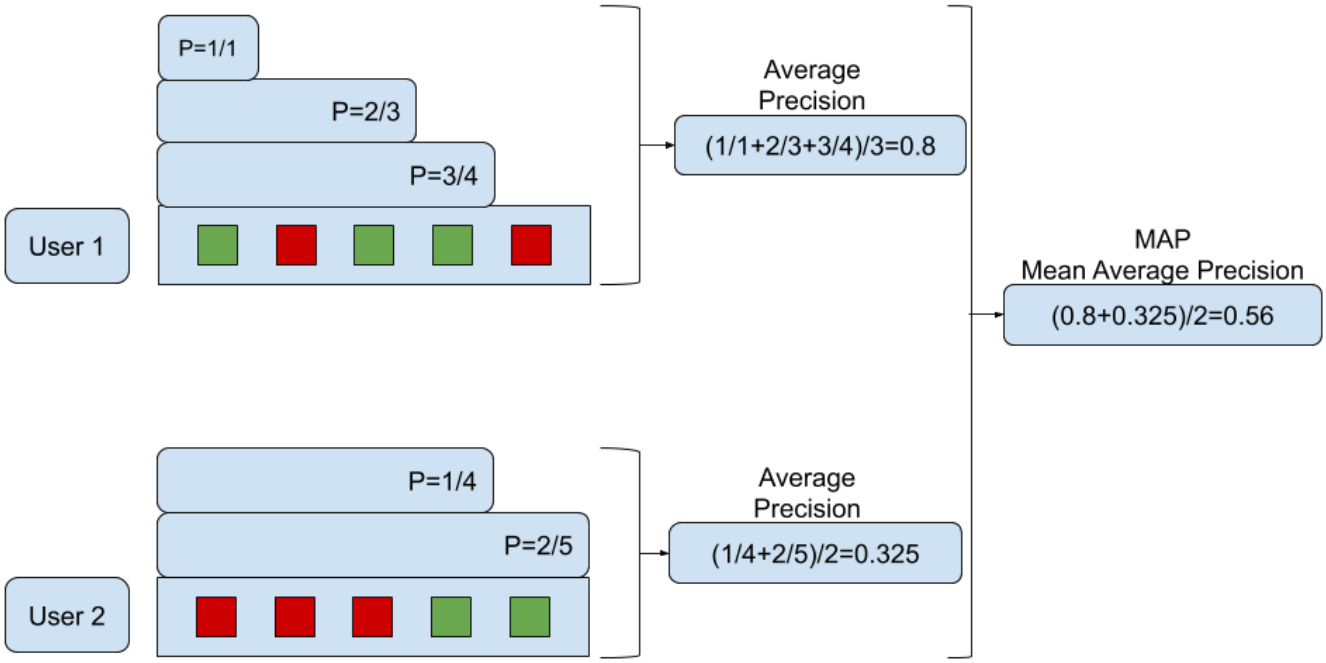
The Average Prediction (AP) metric tries to approximate this weighting sliding scale. It uses a combination of the precision at successive sub-lists, combined with the change in recall in these sub-lists. The calculation goes as follows:

For each user
- For each relevant item
  - Compute precision of list *through* that item
- Average sub-list precisions

MAP calculation algorithm

Here is a diagram to help with visualizing the process:

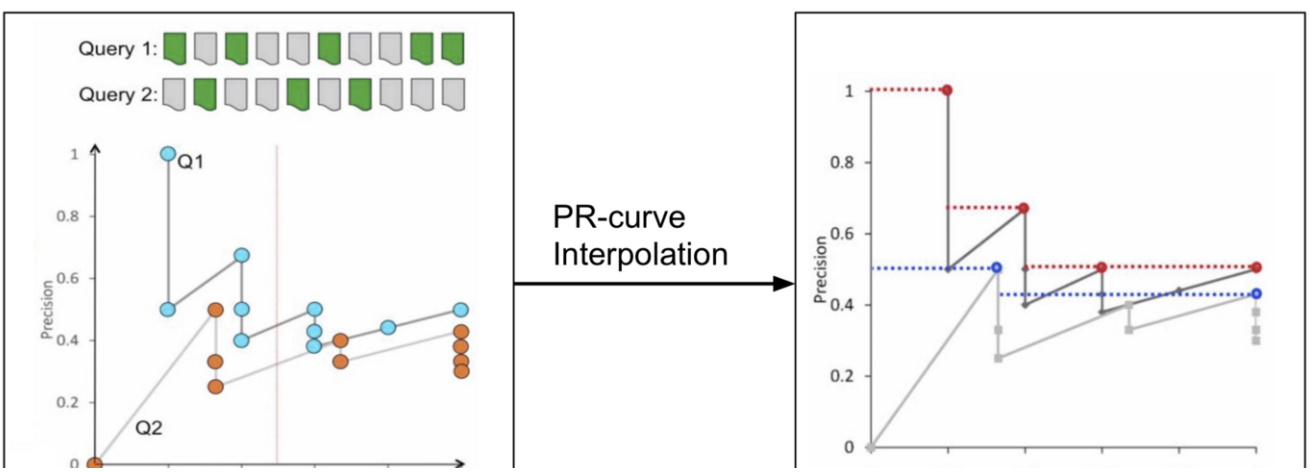
Relevant Item
Non- Relevant Item

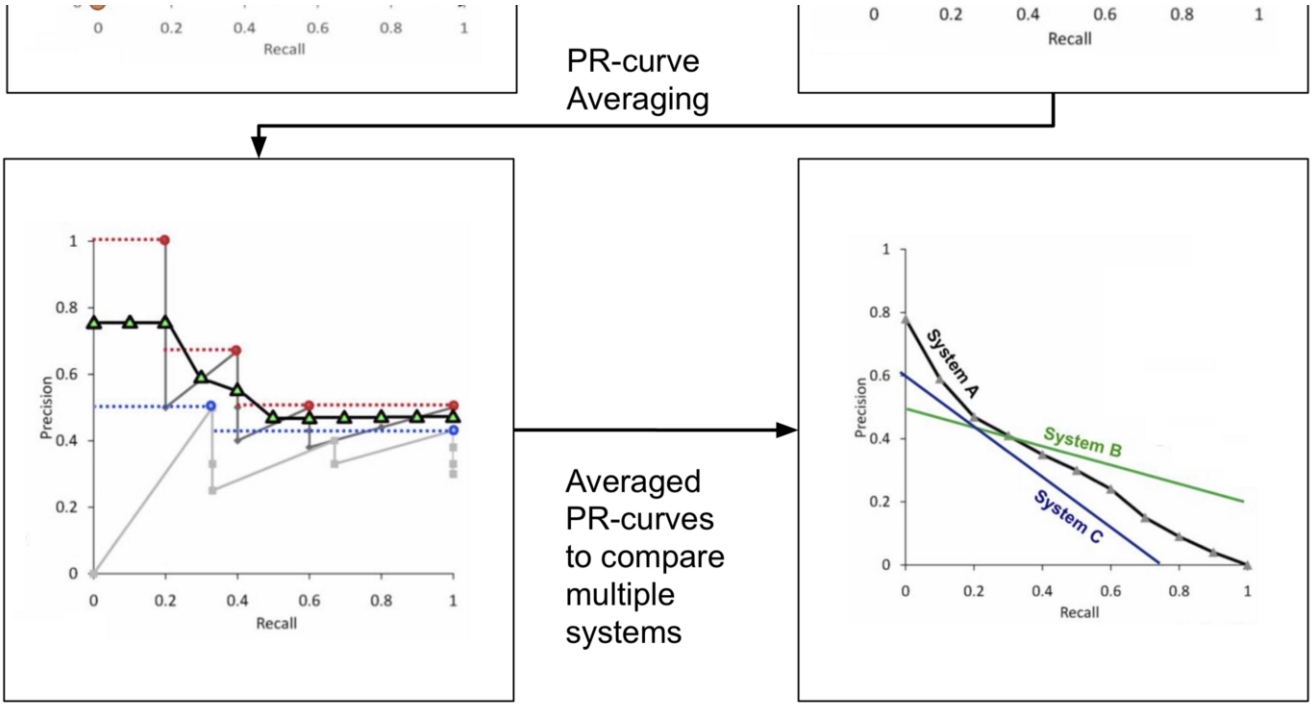


Example MAP calculation

From the figure above, we see that the Average Precision metric is at the single recommendation list, i.e. user level. Computing the precision through this item means sub-dividing the recommendation list. We examine a new sub-list every time we get a relevant item. Then we calculate the precision on this current sublist. We do this for every sublist until we reach the end of our recommendations. Now that we have a set of precisions, we average them to get the average precision for a single user. Then we get the AP for all users and get the mean average precision.

This is primarily an approximation of the original goal of the AP metric. The AP metric represents the area under the precision-recall curve. We get the precision-recall curve by computing the precision as a function of recall values. In this excellent lecture, the concept is expanded in great detail. Very recommended. The overall process is to generate a PR curve for every user recommended list. Then generate an interpolated PR curve, and finally average the interpolated PR curves. This is the process visually:





Interpretation of the MAP measure through the area under the PR curve

To compare two systems we want the largest possible area under the PR curve. In the above example we compare systems A, B and C. We notice that system A is better than system C for all levels of recall. However, system A and B intersect where system B does better at higher levels of recall. The problem with this scenario is that it is hard to determine which system does better overall. Plots are harder to interpret than single metrics. This is why researchers came up with a single metric to approximate the Average Precision (i.e. area under the precision-recall curve). Here is my annotated approximation I adapted from the wikipedia page that describes this process:

Area under the PR-Curve

$$\text{AveP} = \int_0^1 p(r)dr$$

Finite sum approximation over the every ranked recommendations

$$\text{AveP} = \sum_{k=1}^{n} P(k)\Delta r(k)$$

n: number of retrieved items

$\Delta r(k)$: change in recall from items k to k-1

$P(k)$: precision at cut-off k in the list

k: rank in the sequence of retrieved items

Simplified to

$$\text{AveP} = \frac{\sum_{k=1}^{n}(P(k) \times \text{rel}(k))}{\text{number of relevant documents}}$$

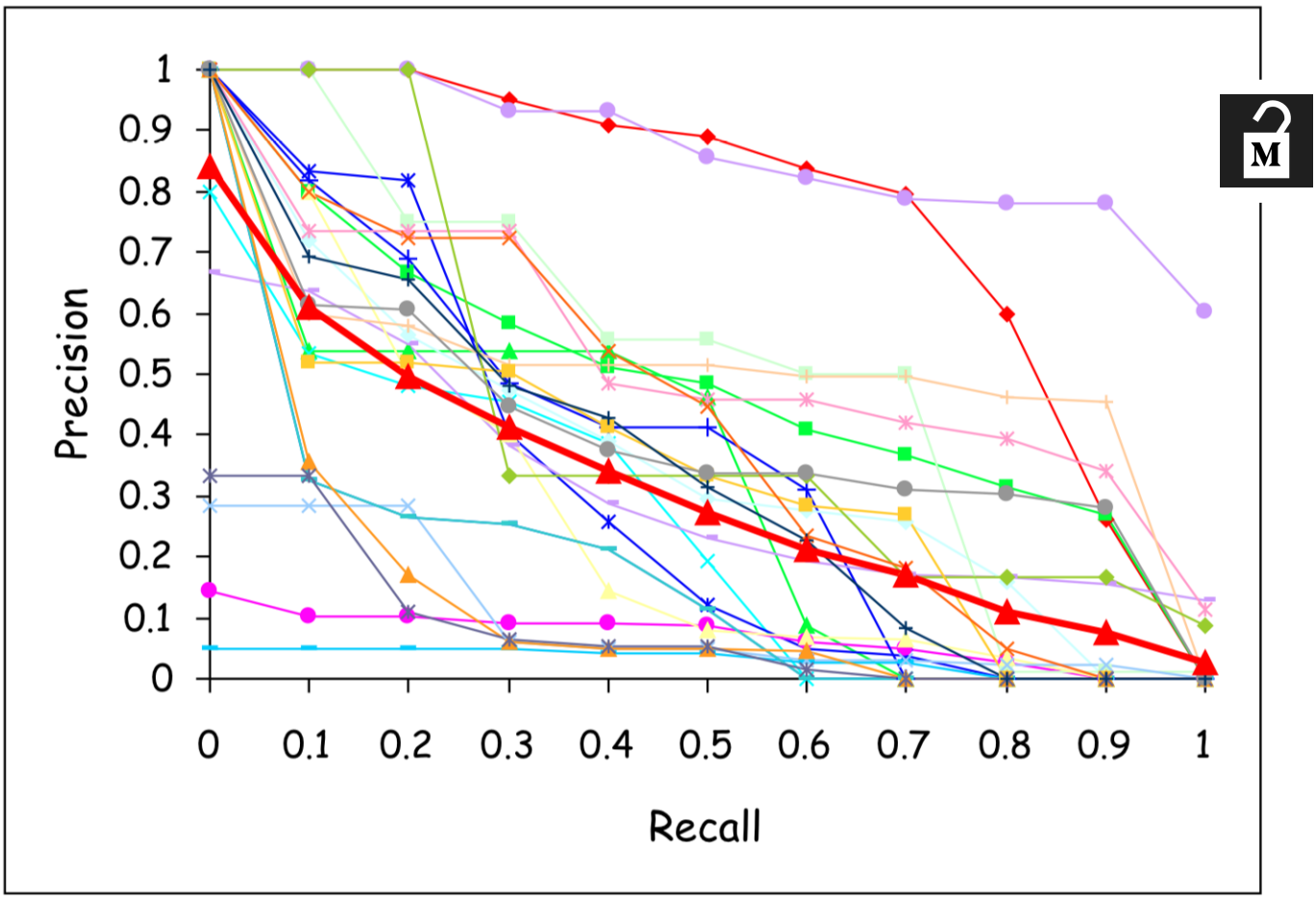
$\text{rel}(k)$: indicator function equaling 1 if the item at rank k is a relevant item

In code to return a vector of average precision scores for every k-th element. true_positive is an ordered list of relevancy of recommended items

```
tp_cumsum = np.cumsum(true_positive)
val_counter = np.cumsum(np.ones(len(true_positive)))
return np.cumsum(tp_cumsum * true_positive / val_counter) / tp_cumsum
```

Annotated derivation of the discrete MAP metric

One last point is realizing what we are actually averaging. This means averaging noisy signals across many users. Below is a plot of the noise that is common across many users. This presentation goes in more details about this issue. This concern is useful to keep in mind when interpreting the MAP score. In the plot below we can see the bright red line is the average PR-curve. The other individual curves in the plot below are for each user for a list of N users. Indeed effectiveness can vary widely across queries. The **MAP averaging will undoubtedly have an effect on the reported performance**. Such sample curves can help evaluate the quality of the MAP metric.



Demonstration of the impact of averaging of the MAP measure across many users.

I invite you to take a look at further writings around the meaning of the PR-curve. The following works here and here provide nice deep dives into the MAP metric.

MAP Pros

- Gives a single metric that represents the complex Area under the Precision-Recall curve. This provides the average precision per list.

- Handles the **ranking of lists** recommended items naturally. This is in contrast to metrics that considering the retrieved items as **sets.**

- This metric is able to give more weight to errors that happen high up in the recommended lists. Conversely, it gives less weight to errors that happens deeper in the recommended lists. This matches the need to show as many relevant items as possible high up the recommended list.

MAP Cons

- This metrics shines for binary (relevant/non-relevant) ratings. However, it is not fit for fine-grained numerical ratings. This metric is unable to extract an error measure from this information.

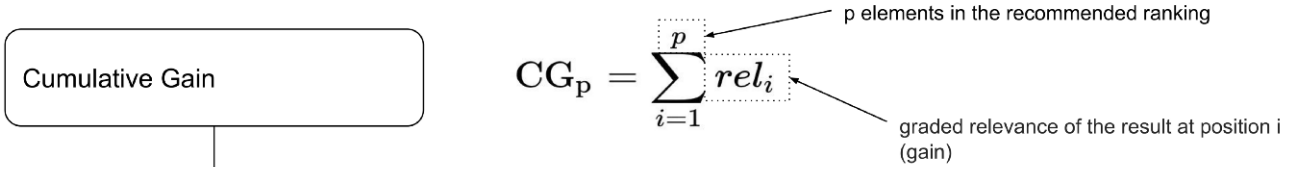
- With fine-grained ratings, for example on a scale from 1 to 5 stars, the evaluation would need first to threshold the ratings to make binary relevancies. One optio to consider only ratings bigger than 4 as relevant. This introduces bias in the evaluation metric because of the manual threshold. Besides, we are throwing a way the fine-grained information. This information is in the difference between a 4 and 5 stars ratings, as well as the information in the non-relevant items. Is a 1 star rating really the same as a 3 stars rating?

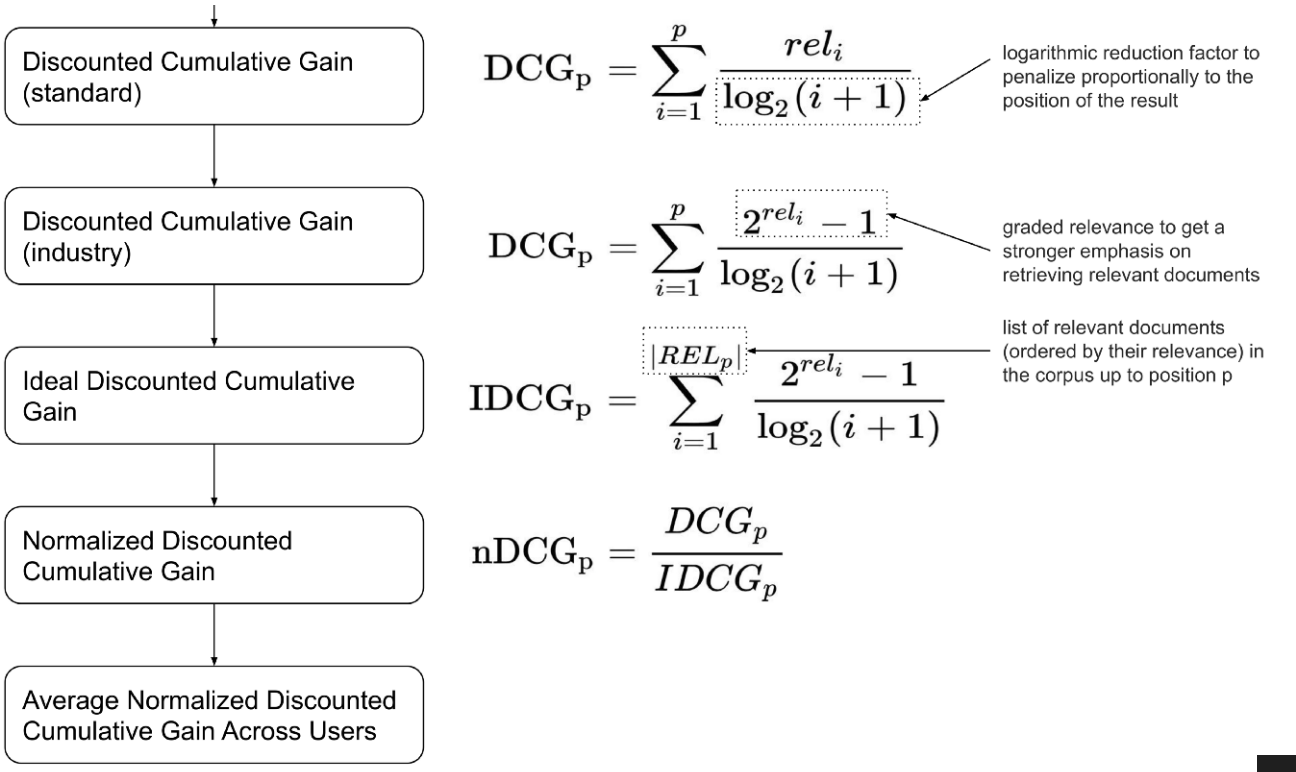
To deal with these issues the recsys community has come up with another more recent metric. This metric takes into account the fined grained information included in the ratings. Let's take a look at the Normalized Discounted Cumulative Gain (NDCG) metric.

Normalized Discounted Cumulative Gain

The goal of the MAP measure is similar to the goal of the NDCG metric. They both value putting highly relevant documents high up the recommended lists. However, the NDCG further tunes the recommended lists evaluation. It is able to use the fact that some documents are "more" relevant than others. Highly relevant items should come before medium relevant items, which should come before non-relevant items.

I provide the following annotated diagram that shows the stages of calculating the NDCG linearly:

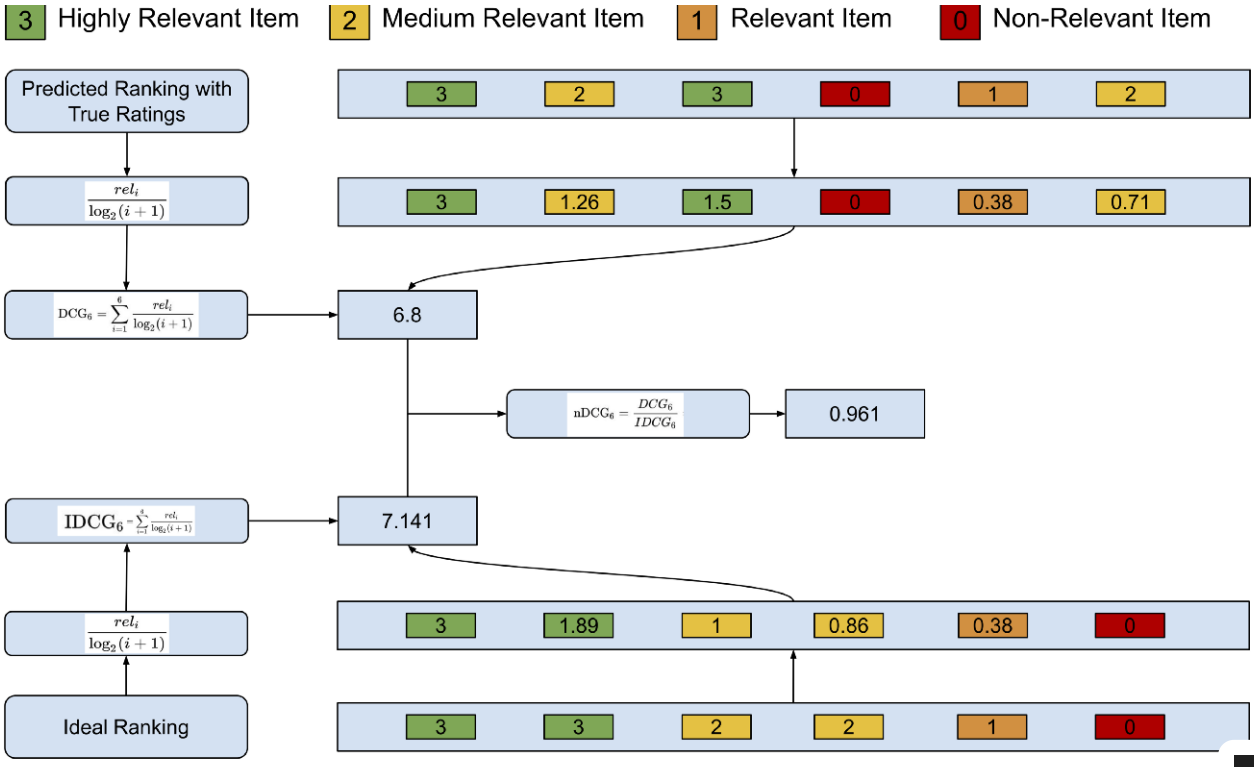




Annotated derivation of the NDCG metric

Before the NDCG we had the cumulative gain CG. This represented a basic measure to accumulate the graded relevances. This metric does not take into account the position of the elements in the ranked list. For ranking tasks, we need to increase the relative impact of the position of elements in the ranked list. The standard Discounted Cumulative Gain, DCG, adds a logarithmic reduction factor to penalize the relevance score proportionally to the position of the item. Furthermore, in industrial applications, it is common to see that the relevance scores get a boost to emphasis retrieving relevant documents. This appears in the industrial DCG formula.

We are dealing with dynamic systems. Users will get a variable number of relevant items recommended. This makes the DCG measure not comparable across users. We need to normalize the metric to be between 0 and 1. To this effect, we determine the ideal ranking for a user. Then we use that ranking as the Ideal Discounted Cumulative Gain IDCG. This provides a nice normalization factor. It helps compute the Normalized Discounted Cumulative Gain. As this is a per-user metric, we need to calculate this metric for all users in the test set. Then we average across users to get a single number. This average is then used for comparing recsys systems to each other. To visualize this process, we go through the calculation in the figure below with the predicted and ideal ranking for a single user.



NDCG Pros

- The primary advantage of the NDCG is that it takes into account the graded relevance values. When they are available in the dataset, the NDCG is a good fit.
- Compared to the MAP metric it does a good job at evaluating the position of ranked items. It operates beyond the binary relevant/non-relevant scenario.
- The smooth logarithmic discounting factor has a good theoretical basis discussed here. The authors of that work show that for every pair of substantially different ranking recommender, the NDCG metric is consistently able to determine the better one.

NDCG Cons

- The NDCG has some issues with partial feedback. This happens when we have incomplete ratings. This is case for the majority of recommender systems situations. If we had complete ratings there would be no real task to achieve! In this case, the recsys system owner needs to decide how to impute the missing ratings. Setting the missing values to 0 would mark them as irrelevant items. Other calculated value such as the mean/median rating for a user can also help with this drawback.
- Next, the user needs to manually handle the case where the IDCG is equal to zero. This occurs when users have no relevant documents. A strategy here is to set the NDCG to 0 as well.

- Another issue is handling NDCG@K. The size of the ranked list returned by the recsys system can be less than K. To handle this we can consider fixed-size result sets and pad the smaller sets with minimum scores.

As I said the primary advantage of the NDCG is that it takes into account the graded relevance values. If your dataset has the right form and you are dealing with graded relevance, then NDCG measure is your go-to metric.

That's all folks

Understanding metrics used for machine learning (ML) systems is important. ML practitioners invest signification budgets to move prototypes from research to production and offline metrics are crucial indicators for promoting a new model to production. Understanding the drawbacks of each metrics helps build personal credibility and helps avoid the trap of prematurely proclaiming victory. Reporting small improvements on inadequate metrics is a well known ML trap.

I hope this post helped you explore the three metrics we discussed and expand your ML toolbox.

One last thing! All this complexity is probably making your ML code harder to maintain and evolve:

Checkout my latest (in progress) book about topics surrounding this blog post with code examples, in depth discussions, and more to help with that!

https://leanpub.com/cleanmachinelearningcode

References

- Rank-aware recsys metrics: Rank-Aware Top-N Metrics
- Evaluation 11 Interpolated recall-precision plot: https://www.youtube.com/watch?v=yjCMEjoc_ZI
- Code for AP calculation: https://github.com/krzjoa/kaggle-metrics/blob/master/kaggle_metrics/order_based.py
- Stanford Course slides on rank-ware metrics: https://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-6-per.pdf

- https://web.stanford.edu/class/cs276/handouts/
- Evaluating Retrieval System Effectiveness
- http://www.cs.utexas.edu/~mooney/ir-course/slides/Evaluation.ppt
- http://www.nii.ac.jp/TechReports/05-014E.pdf
- http://hal.archives-ouvertes.fr/docs/00/72/67/60/PDF/07-busa-fekete.pdf

Machine Learning Metrics Recommendation System Data Science